RESUME

1. Tentang version control

Version control adalah sebuah sistem yang merekam perubahan-perubahan dari sebuah berkas atau sekumpulan berkas dari waktu ke waktu sehingga Anda dapat menilik kembali versi khusus suatu saat nanti. Sebagai contoh, pada buku ini Anda akan menggunakan sumber kode perangkat lunak sebagai berkas-berkas yang direkam dengan version control, walau pada kenyataannya Anda dapat melakukan ini dengan hampir semua jenis berkas pada komputer.

Version Control System (VCS) adalah hal yang bijak untuk digunakan. VCS memperbolehkan Anda untuk mengembalikan berkas-berkas ke keadaan sebelumnya, mengembalikan seluruh proyek kembali ke keadaan sebelumnya, membandingkan perubahan-perubahan di setiap waktu, melihat siapa yang terakhir mengubah sesuatu yang mungkin menimbulkan masalah, siapa dan kapan yang mengenalkan sebuah isu dan banyak lagi. Menggunakan VCS secara umum juga berarti bahwa jika Anda melakukan kesalahan atau menghilangkan berkas, Anda dapat dengan mudah memulihkannya. Sebagai tambahan, Anda mendapatkan semua ini dengan biaya yang sangat sedikit.

1. Jenis jenis version control

Jenis-jenis *version control system* yang perlu diketahui adalah seperti berikut ini:

1. *Local version control systems*

*local version control systems* merupakan jenis VCS paling sederhana dan memiliki *database* yang menyimpan semua perubahan pada *file*. LCS juga merupakan salah satu alat VCS yang paling umum. Sistem ini mampu membuat *set patch,* atau perbedaan antara *file*, dalam format khusus pada sebuah *disk*. Dengan menambahkan semua *patch*, LCS dapat menciptakan bentuk orisinal dari semua *file* ketika seorang *engineer* atau *developer* membutuhkannya.

1. *Centralized version control systems*

Jenis *version control system* berikutnya adalah CVCS. Sistem satu ini kerap menjadi jembatan kolaborasi di antara para *engineer* dan *developer*. CVCS dapat memberikan informasi tentang apa yang dilakukan para *engineer* dan *developer* pada sebuah proyek. Hal ini memungkinkan *database administrator* untuk mengontrol pekerjaan yang harus dilakukan seluruh pihak dalam proyek tanpa terjadinya benturan dan miskomunikasi.

1. *Distributed version control systems*

DVCS mengandung banyak repositori. Sistem ini mendukung cara kerja di mana tiap pengguna memiliki repositori dan *copy* dari pekerjaan mereka sendiri. Ketika pengguna melakukan perubahan, DVCS tidak akan memberi orang lain akses ke pada perubahan yang telah dibuat tersebut. Hal ini karena DVCS akan mencerminkan perubahan tersebut dalam repositori masing-masing pengguna dan mereka perlu mendorongnya agar terlihat pada repositori pusat. Intinya, DVCS mendorong budaya tanggung jawab pada tiap perubahan yang dilakukan masing-masing *engineer* dan *developer*.

1. Jelaskan bedanya
2. Sejarah git

Bersamaan dengan banyak hal besar dalam hidup, Git dimulai dengan sedikit kehancuran kreatifitas dan pertentangan yang ganas.Kernel Linux adalah proyek perangkat lunak sumber terbuka dalam lingkup yang cukup besar. Sebagian besar waktu pemeliharaan dari kernel Linux (1991-2002), perubahan-perubahan pada perangkat lunak diberikan sebagai patch dan berkas terarsipkan. Pada 2002, proyek kernel Linux mulai menggunakan DVCS terpatenkan bernama BitKeeper.

Pada 2005, hubungan antara komunitas yang mengembangkan kernel Linux dan perusahaan komersil yang mengembangkan BitKeeper terputus, dan status bebas biaya dari alatnya dicabut. Hal ini mendesak komunitas pengembangan Linux (khususnya Linus Torvalds, pencipta Linux) untuk mengembangkan alat mereka sendiri berdasarkan beberapa pelajaran yang telah mereka pelajari ketika menggunakan BitKeeper. Beberapa sasaran dari sistem baru tersebut adalah sebagai berikut:

* Kecepatan
* Rancangan yang sederhana
* Dukungan yang kuat untuk pengembangan non-linier (ribuan cabang paralel)
* Benar-benar tersebar
* Mampu menangani proyek besar seperti Linux secara efisien (kecepatan dan ukuran data)

Sejak kelahirannya pada 2005, Git telah berevolusi dan berkembang untuk dapat digunakan dengan mudah namun tetap memiliki kualitas awal tersebut. Git sangat cepat, sangat efisien dengan proyek-proyek besar, dan Git memiliki sistem percabangan yang hebat untuk pengembangan non-linear

1. Command dalam git

## Perintah dasar GIT

* **git config** Salah satu perintah git yang paling banyak digunakan adalah **git config**, yang bisa digunakan untuk mengatur konfigurasi tertentu sesuai keinginan pengguna, seperti email, algoritma untuk diff, username, format file, dll. Contohnya, perintah berikut bisa digunakan untuk mengatur email: git config --global user.email sam@google.com
* **git init** Perintah ini digunakan untuk membuat repositori baru. Caranya: git init
* **git add** Perintah git add bisa digunakan untuk menambahkan file ke index. Contohnya, perintah berikut ii akan menambahkan file bernama temp.txt yang ada di direktori lokal ke index: git add temp.txt
* **git clone** Perintah git clone digunakan untuk checkout repositori. Jia repositori berada di remove server, gunakan: git clone [alex@93.188.160.58:/path/to/repository](mailto:alex@93.188.160.58:/path/to/repository) Jika salinan repositori lokal ingin dibuat, gunakan: git clone /path/to/repository
* **git commit** Perintah git commit digunakan untuk melakukan commit pada perubahan ke head. Ingat bahwa perubahan apapun yang di-commit tidak akan langsung ke remote repository. Gunakan: git commit –m “Isi dengan keterangan untuk commit”
* **git status**  
  Perintah git status menampilkan daftar file yang berubah bersama dengan file yang ingin di tambahkan atau di-commit. Gunakan: git status
* **git push** git push adalah perintah git dasar lainnya. Push akan mengirimkan perubahan ke master branch dari remote repository yang berhubungan dengan direktori kerja Anda. Misalnya: git push origin master
* **git checkout** Perintah git checkout bisa digunakan untuk membuat branch atau untuk berpindah diantaranya. Misalnya, perintah berikut ini akan membuat branch baru dan berpindah ke dalamnya: command git checkout -b <nama-branch> Untuk berpindah dari branch satu ke lainnya, gunakan: git checkout <branch-name>
* **git remote** Perintah git remote akan membuat user terhubung ke remote repository. Perintah berikut ini akan menampilkan repository yang sedang dikonfigurasi: git rmote -v Perintah ini membuat user bisa menghubungkan repository lokal ke remote server: git remote add origin <93.188.160.58>
* **git branch** Perintah git branch bisa digunakan untuk me-list, membuat atau menghapus branch. Untuk menampilkan semua branch yang ada di repository, gunakan: git branch Untuk menghapus branch: git branch -d <branch-name>
* **git pull** Untuk menggabungkan semua perubahan yang ada di remote repository ke direktori lokal, gunakan perintah pull: git pull
* **git merge** Perintah merge digunakan untuk menggabungkan sebuah branch ke branch aktif. Gunakan: git merge <nama-branch>
* **git diff** Perintah git diff digunakan untuk menampilkan conflicts. Untuk melihat conflicts dengan file dasar, gunakan: git diff --base <nama-file> Perintah berikut digunakan untuk menampilkan conflicts diantara branch yang akan di-merge: git diff <source-branch> <target-branch> Untuk menampilkan semau conflict yang ada, gunakan: git diff
* **git tag** Tagging digunakan untuk menandai commits tertentu. Contohnya: git tag 1.1.0 <insert-commitID-here>
* **git log** Dengan menjalankan peritah ini akan menampilkan daftar commits yang ada di branch beserta detail-nya. Contoh outputnya adalah: commit 15f4b6c44b3c8344caasdac9e4be13246e21sadw Author: Alex Hunter <alexh@gmail.com> Date: Mon Oct 1 12:56:29 2016 -0600
* **git reset** Untuk me-reset index dan bekerja dengan kondisi commit paling baru, gunakan perintah git reset: git reset --hard HEAD
* **git rm** Gunakan perintah ini untuk menghapus file dari index dan direktori kerja. Contohnya: git rm filename.txt
* **git stash** Mungkin inilah salah satu perintah dasar git yang jarang digunakan orang, yang bisa membantu menyimpan perubahan yang tidak langsung di-commit, namun hanya sementara. Contoh: git stash
* **git show** Untuk menampilkan informasi tentang object pada git, gunakan git show: git show
* **git fetch** Perintah ini digunakan untuk menampilkan semua object dari remote repository yang tidak berada di direktori kerja lokal. Contohnya: git fetch origin
* **git ls-tree**Untuk menampilkan susunan object berdasarkan nama dan mode setiap item, dan nilai blob SHA-1, gunakan perintah git ls-tree. Contohnya: git ls-tree HEAD
* **git cat-file**Menggunakan nilai SHA-1, menampilkan tipe object dengan menggunakan perintah git cat-file. Contohnya: git cat-file –p d670460b4b4aece5915caf5c68d12f560a9fe3e4
* **git prep** git prep mengizinkan pengguna mencari frase dan/atau kata yang berada di dalam direktori. Contohnya, untuk mencari www.hostinger.co.id di dalam semua file, gunakan: git grep "www.hostinger.co.id"
* **gitk**  
  gitk adalah tampilan grafis dari repository lokal yang bisa dipanggil dengan menjalankan perintah:

gitk

* **git instaweb** Dengan perintah git instaweb, web server bisa dijalan berdampingan dengan repository lokal. Nantinya web browser akan langsung diarahkan ke server tersebut. Contohnya: git instaweb –httpd=webrick
* **git gc**  
  Untuk mengoptimasi repository melalui garbage collection, yang akan membersihkan file yang tidak dibutuhkan dan mengoptimasinya, gunakan: git gc
* **git archive** Perintah git archive memungkinkan user membuat file zip atau tar yang mengandung susunan repository. Contohnya: git archive --format=tar master
* **git prune** Melalui perintah git prune, object yang tidak memiliki incoming pointers akan dihapus. Gunakan: git prune
* **git fsck** Untuk membuat pengecekan keseluruhan dari file system git, gunakan perintah git fsck. Object yang corrupt akan dikenali: git fsck
* **git rebase** Perintah ini digunakan untuk menerapkan ulang commit di branch yang lain. Contohnya: git rebase master

1. Penjelasan tentang repository git

Repositori Git adalah lokasi penyimpanan pusat untuk mengelola dan melacak perubahan dalam file dan direktori. Ini adalah komponen penting dari sistem kontrol versi Git, yang memungkinkan pengembangan kolaboratif dan memungkinkan banyak pengembang mengerjakan sebuah proyek secara bersamaan. Repositori Git banyak digunakan dalam pengembangan perangkat lunak, yang memfasilitasi manajemen kode yang efisien dan terkontrol.

Repositori Git menyimpan semua versi file dalam suatu proyek, memungkinkan pengembang melacak perubahan, berkolaborasi, dan dengan mudah kembali ke versi sebelumnya jika diperlukan. Setiap perubahan file dicatat sebagai commit, yang disusun menjadi beberapa cabang, memberikan sejarah terstruktur pengembangan proyek. Sistem ini memungkinkan pengembang untuk mengerjakan cabang yang berbeda secara mandiri dan kemudian menggabungkan perubahan mereka kembali ke cabang utama.

1. Flow dari git

Git Flow adalah ide abstrak dari alur kerja Git. Ini membantu pengembangan perangkat lunak berkelanjutan dan penerapan praktik DevOps. **Alur Kerja Git Flow mendefinisikan model percabangan ketat yang dirancang seputar rilis proyek.** Hal ini memberikan kerangka kerja yang kuat untuk mengelola proyek yang lebih besar.

Git Flow cocok untuk proyek yang memiliki siklus rilis terjadwal dan untuk praktik terbaik DevOps dalam pengiriman berkelanjutan. **Ini memberikan peran yang sangat spesifik ke berbagai cabang dan menentukan bagaimana dan kapan mereka harus berinteraksi.** Ia menggunakan cabang individu untuk mempersiapkan, memelihara, dan merekam rilis.

# Cara kerja Git Flow

# 1. Mengembangkan dan Menguasai Cabang

Alih-alih menggunakan satu cabang master, Git Flow menggunakan dua cabang untuk mencatat sejarah proyek. Hal ini didasarkan pada dua cabang utama dengan masa pakai tak terbatas yaitu **master dan development** .

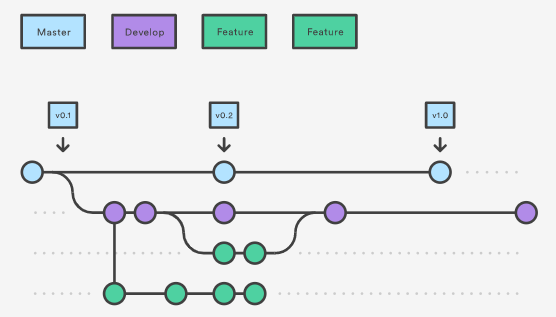
* **Cabang Master:** Cabang master berisi kode produksi dan menyimpan riwayat rilis resmi.
* **Cabang Pengembangan:** Cabang pengembangan berisi kode pra-produksi dan berfungsi sebagai cabang integrasi untuk fitur.

Saat menggunakan pustaka ekstensi git-flow, mengeksekusi git flow init pada repo yang ada akan membuat cabang pengembangan.

# 2. Cabang Fitur

Setiap fitur baru harus berada di cabangnya, yang dapat dikirim ke repositori pusat untuk pencadangan/kolaborasi. Cabang fitur menggunakan pengembangan terbaru **sebagai cabang induknya** . Ketika suatu fitur selesai, fitur tersebut akan **digabungkan kembali menjadi development** . Fitur tidak boleh berinteraksi langsung dengan **cabang master** .

Alur kerja cabang fitur ditunjukkan dalam diagram yang diberikan:



## 2.1. Membuat Cabang Fitur

Tanpa ekstensi git-flow:

* git checkout develop
* git checkout -b feature\_branch

Dengan ekstensi git-flow:

* git flow feature start feature\_branch

## 2.2. Menyelesaikan Cabang Fitur

Tanpa ekstensi git-flow:

* git checkout develop
* git merge feature\_branch

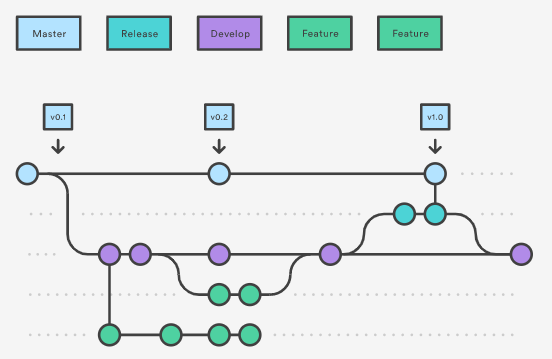
Dengan ekstensi git-flow:

* git flow feature finish feature\_branch

# 3. Cabang Rilis

Setelah pengembangan memperoleh cukup fitur untuk sebuah rilis (atau tanggal rilis yang telah ditentukan semakin dekat), kami **membuat cabang rilis dari pengembangan** . Pembuatan cabang ini **akan memulai siklus rilis berikutnya** , sehingga tidak ada fitur baru yang dapat ditambahkan setelah titik ini — hanya perbaikan bug, pembuatan dokumentasi, dan tugas berorientasi rilis lainnya yang boleh dimasukkan ke dalam cabang ini. Cabang rilis mungkin bercabang dari pengembangan dan **harus digabungkan menjadi master dan pengembangan** .

Alur kerja cabang rilis ditunjukkan dalam diagram yang diberikan:



Setelah cabang rilis siap dikirim, cabang tersebut akan digabungkan menjadi master dan **ditandai dengan nomor versi** . Selain itu, pembaruan tersebut harus **digabungkan kembali ke dalam pengembangan karena pembaruan penting mungkin telah ditambahkan** ke cabang rilis dan pembaruan tersebut harus dapat diakses ke fitur-fitur baru. Jadi, setelah rilis siap dikirim, rilis tersebut akan digabungkan menjadi master dan dikembangkan, kemudian **cabang rilis akan dihapus** .

Menggunakan cabang khusus untuk menyiapkan rilis memungkinkan satu tim menyempurnakan rilis saat ini sementara tim lain terus mengerjakan fitur untuk rilis berikutnya.

## 3.1. Membuat Cabang Rilis

Tanpa ekstensi git-flow:

* git checkout develop
* git checkout -b release/0.1.0

Saat menggunakan ekstensi git-flow:

* git flow release start 0.1.0
* Switched to a new branch 'release/0.1.0'

## 3.2. Menyelesaikan Cabang Rilis

Tanpa ekstensi git-flow:

* git checkout master
* git merge release/0.1.0

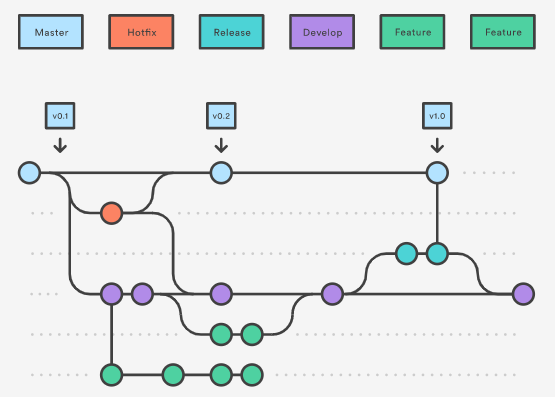
Dengan ekstensi git-flow:

* git flow release finish 0.1.0

# 4. Cabang Perbaikan Terbaru

Cabang pemeliharaan atau “perbaikan terbaru” **digunakan untuk menambal rilis produksi dengan cepat** . Cabang perbaikan terbaru diperlukan untuk segera bertindak atas status master yang tidak diinginkan. Cabang perbaikan terbaru sangat mirip dengan cabang rilis dan cabang fitur kecuali cabang tersebut didasarkan pada master, bukan pengembangan. Ini adalah **satu-satunya cabang yang harus bercabang langsung dari master** . Segera setelah perbaikan selesai, perbaikan tersebut harus **digabungkan menjadi master dan development (atau cabang rilis saat ini)** , dan **cabang master harus ditandai dengan nomor versi yang diperbarui** .

Alur kerja cabang perbaikan terbaru ditunjukkan dalam diagram yang diberikan:



Memiliki jalur pengembangan khusus untuk perbaikan bug **memungkinkan tim Anda mengatasi masalah tanpa mengganggu alur kerja lainnya** atau menunggu siklus rilis berikutnya.

## 4.1. Membuat Cabang Perbaikan Terbaru

Tanpa ekstensi git-flow:

* git checkout master
* git checkout -b hotfix\_branch

Dengan ekstensi git-flow:

* git flow hotfix start hotfix\_branch

## 4.2. Menyelesaikan Cabang Perbaikan Terbaru

Tanpa ekstensi git-flow:

* git checkout master
* git merge hotfix\_branch
* git checkout develop
* git merge hotfix\_branch

Dengan ekstensi git-flow:

* git branch -D hotfix\_branch
* git flow hotfix finish hotfix\_branch

# Keuntungan dari Git Flow

Sekarang mari kita bahas ringkasan keuntungan utama yang disediakan oleh Git flow:

* Memastikan keadaan cabang yang bersih pada saat tertentu dalam siklus hidup suatu proyek
* Konvensi penamaan cabang mengikuti pola sistematis sehingga lebih mudah dipahami
* Memiliki ekstensi dan dukungan pada alat git yang paling banyak digunakan
* Ideal jika mempertahankan beberapa versi dalam produksi
* Cocok untuk alur kerja perangkat lunak berbasis rilis.
* Menawarkan saluran khusus untuk perbaikan terbaru pada produksi.

# Kekurangan Aliran Git

Tidak ada yang ideal, jadi Git Flow juga memiliki beberapa kelemahan seperti:

* Riwayat Git menjadi tidak dapat dibaca
* Pemisahan cabang master/pengembangan dianggap mubazir dan mempersulit Pengiriman/Integrasi Berkelanjutan
* Tidak disarankan jika mempertahankan satu versi dalam produksi

https://github.com/rikaalii/rikaalii.git